บทที่ 6
ภาษาสอบถามเชิงโครงสร้าง
(Structured Query Language :SQL)

Database Language (ภาษาฐานข้อมูล)ในแนวความคิดนั้นต้องมีลักษณะดังนี้

1. ให้ผู้ใช้สามารถสร้างโครงสร้างของ ฐานข้อมูล และ Table ได้
2. ให้ผู้ใช้สามารถทำ Data Management ได้ (Add, Delete + Modify)
3. ให้ผู้ใช้สามารถออกแบบ Query ที่รับซับซ้อนเพื่อเปลี่ยน Raw Data → Information ที่ต้องการได้

   (Information : คือ ข้อมูลที่ผ่านการประมวลผลแล้ว หรือก็คือ Output Data ที่สามารถนำมาใช้ในการตัดสินใจได้)
4. ต้องทำ Basic Function (การทำงานพื้นฐาน) โดยผู้ใช้ต้องไม่ยุ่งยาก
5. โครงสร้างของคำสั่ง และ รูปแบบการใช้ ต้องเรียนรู้ง่าย
6. ต้องเป็น Portability หมายความว่า สามารถเคลื่อนย้ายได้ คือสามารถใช้ได้กับเครื่องหลายๆ เครื่อง ไม่ใช่ใช้กับเครื่องใดเครื่องหนึ่ง

   นั่นคือ หลักการพื้นฐานที่ภาษา Query ควรจะมีอย่างเป็นมาตรฐาน เพื่อที่ผู้ใช้จะสามารถใช้ RDBMS ได้อย่างสะดวก เมื่อต้องเปลี่ยนจาก RDBMS ตัวหนึ่งไปยังอีกตัวหนึ่ง

SQL เป็น The Basic of The Relational Model's Standard Language

- มีประมาณ 30 คำสั่ง
- ออกแบบมาเพื่อทำงานกับ Application ต่างๆ ใน Relational database
- มีศัพท์ที่ใช้จำกัด จึงง่ายต่อการเรียนรู้
- เป็นภาษาที่ใช้ง่าย แต่การทำงานที่เกิดขึ้นนั้นให้ผลที่ผู้ใช้ต้องการได้ในรูปแบบต่างๆ ซึ่งถึงแม้จะซับซ้อนก็ทำได้
- เป็น Non-Procedural Language คือผู้ใช้ระบุแต่เพียง "What"
- ผู้ใช้ไม่จำเป็นต้องรู้โครงสร้างหรือรูปแบบของข้อมูล
- สามารถนำไปใช้กับ GUI-Base S.W. หรือ Special Add-On Utility เพื่อให้เกิดการทำงานที่ดีขึ้น

CT 316 (S)

ทำงานที่ดีขึ้น

SQL มีลักษณะที่ตรงกับความต้องการที่ควรมีใน Database Language เพราะ

1. SQL ครอบคลุมถึงการทำงานในหัวข้อต่อไปนี้
   - Data Definition - สร้างโครงสร้างของ ฐานข้อมูล และTable
   - Data Management – ใช้ชุดของคำสั่งในการ Update ข้อมูลในตารางได้ ได้แก่ การเพิ่มข้อมูล (Add), การลบ (Delete), การปรับปรุงแก้ไข (Modify)
   - Data Query - ใช้ชุดของคำสั่งในการสำรวจข้อมูลใน ฐานข้อมูล และยังให้ผู้ใช้เปลี่ยน Raw Data เป็น Information ได้
2. SQL เรียนรู้ได้ง่าย คือสามารถทำงานได้ทั้ง 3 หัวข้อโดยใช้คำศัพท์ต่ำกว่า 100 คำที่ยังเป็น Non-Procedural Language ด้วย หมายความว่า เป็นภาษาที่ผู้ใช้เพียงแค่ระบุถึงสิ่งที่ต้องการ แต่ไม่ต้องระบุวิธี
3. The American National Standard Institute (ANSI) ได้กำหนด Standard SQL ทำให้แนวทางการใช้ SQL เป็นไปอย่างมีมาตรฐาน (ถึงแม้จะทำให้การใช้งานถูกจำกัดลงและการเปลี่ยนจาก RDBMS หนึ่งไปยัง RDBMS อีกตัวหนึ่งจะยังคงต้องมีการปรับเปลี่ยนอยู่ดี)

6.1 Data Definition Commands (DDC) and Data Definition Language (DDL)

ใช้ในการสร้างโครงสร้างของ ฐานข้อมูล และ Table

ข้อเสีย - DDL ในแต่ละ RDBMS จะแตกต่างกัน

ข้อดี - DDC ใช้ง่าย และใช้สร้าง TABLE ที่ซับซ้อนได้ไม่ว่าจะใช้ RDBMS ใดก็ตาม

1. การสร้าง Data Structure เช่น

```
CREATE DATABASE <DATABASE NAME>
: คำสั่งใช้ในการกำหนดโครงสร้างของ ฐานข้อมูล
สวนคำสั่งที่ใช้ในการสร้าง Table คือ คำสั่ง
CREATE TABLE <TABLE NAME> (
< ATTRIBUTE-NAME      ATTRIBUTE-CHARACTERISTIC,
:
<ATTRIBUTE-NAME       ATTRIBUTE-CHARACTERISTIC,
PRIMARY KEY DESIGNATION,
FOREIGN KEY DESIGNATION AND REQUIREMENT >);
```

รูปแบบของ ANSI SQL (รูปแบบมาตรฐานของ SQL

2.การสร้าง Table Structure

Product

P-CODE	P-DESCRIPTION	P-ONHAND	P-MIN	P-PRICE	V-CODE
11QER/31	HAMMER 12 BL	20	5	800.00	25598
23114-AA	METAL SCREEN	172	80	35.00	21225

Vendor

V-CODE	V-NAME	V-CONTACT	V-PHONE	V-CITY	V-ORDER
21225	BRYSON INC.	SMITH	6/5-223-2233	TX.	Y
21226					
:					

TABLE

CREATE < TABLE NAME > (

< ATTRIBUTE 1 – NAME AND ATTRIBUTE1-CHARACTERISTICS,

:

PRIMARY KEY DESIGNATION,

FOREGN KEY DESIGNATION AND FOREGN KEY REQUIREMINTS >)

EX CREATE TABLE VENDOR (

V-CODE INTEGER NOT NULL UNIQUE, 21225

V-NAME VCHAR(35) NOT NULL,

V-CONTACT VCHAR(15) NOT NULL,

V-PHONE FCHAR(12) NOT NULL,

V-CITY FCHAR(2) NOT NULL,

V-ODER FCHAR(1) NOT NULL,

PRIMARY KEY (V-CODE))..

```
CREATE      TABLE PRODUCT (
    P-CODE      VCHAR(10)          NOT NULL    UNIQUE,
    P-DESCRIPT  VCHAR(30)          NOT NULL,
    P-ONHAND    SMALLINT           NOT NULL,
    P-MIN       1MAILINT           NOT NULL,
    P-PRICE     DECIMAL(8,2) NOT NULL,
    V-CODE      INTEGER,
    PRIMARY KEY (P-CODE),
    FOREIGN  KEY (V-CODE) REFERENCE VENDOR
        ON DELETE RESTRICT
        ON UPDATE CRSCADE ) ;
```

จาก SQL ของการ Create Table เราพบว่า

- Primary Key จะถูกกำหนดให้เป็นทั้ง Unique และ Not Null ที่เป็นเช่นนี้ก็เพื่อต้องการสนับสนุนในเรื่อง Entity Integrity (ความมั่นคงของ Entity (Record))
- การกำหนด Not Null ใน Attribute อื่นๆ ก็เพื่อให้แน่ใจว่าข้อมูลจะต้องป้อนเข้าไปเสมอ ไม่มีการกระโดดข้าม
- รายละเอียดเกี่ยวกับ Attribute จะถูกกำหนดไว้ภายในเครื่องหมาย ( )
- มีการใช้เครื่องหมาย ',' คั่นแต่ละ Attribute แต่ละตัว
- มีการใช้เครื่องหมาย ';' จบคำสั่ง (บาง อาจจะไม่ใช้หรือใช้อย่างอื่น)
- การกำหนด Primary Key ในคำสั่ง เท่ากับเป็นการสนับสนุนในเรื่อง Entity Integrity อีกทั้งยังกำหนดให้เป็น Unique และ Not Null ซึ่งเพิ่มการเป็นเรื่อง Entity Integrity
- การกำหนด Foreign Key เท่ากับเป็นการสนับสนุนในเรื่อง Referential Integrity นอกจากนี้การกำหนด On Delete Restrict และ On Update Cascade ยังเป็นการควบคุมในเรื่องการลบข้อมูลของ Vendor ก็มีการอ้างถึงจากตาราง Product ด้วย นอกจากนั้นการเปลี่ยนแลงใดๆ ที่เกิดขึ้นใน Vendor Table ที่เกี่ยวกับ V-CODE ใดก็ตามจะต้อง ได้รับการแก้ไขใน Product Table ด้วย

6.2 กลุ่มของภาษา DML (Basic Data Management Command)

คำสั่งที่เกี่ยวข้องได้แก่

Insert : เป็นคำสั่งที่ใช้ในการเพิ่มข้อมูลเข้าไปใน Table โดยเพิ่มที่ละ Row อาจเพิ่มจากการที่ Table ไม่มีข้อมูลเลยหรือมีข้อมูลแล้วก็ตาม

Select : เป็นคำสั่งที่ใช้แสดงรายละเอียดของข้อมูลใน Table

Commit : เป็นคำสั่งที่ใช้ Save ข้อมูลลงแผ่น Disk

Update : เป็นคำสั่งที่ใช้เปลี่ยนแปลงข้อมูลใน Table

Delete : เป็นคำสั่งที่ใช้ลบข้อมูลออกจาก Table

Rollback : เป็นคำสั่งที่ทำให้เกิดการ Restore ข้อมูลใน Table ให้อยู่ในสภาพเดิม (ให้ย้อนกลับไปคืนสภาพเดิม จุดเดิมก่อนการเปลี่ยนแปลง การเปลี่ยนแปลง เช่น ไฟฟ้าดับ ไฟฟ้าตก)

Data Entry

การที่ Product Table อ้างถึง V-CODE ซึ่งอยู่ใน Vendor Table เป็น Foreign Key ดังนั้นเราต้องกำหนด Vendor Table ก่อน Product Table นอกจากนี้การใส่ข้อมูลเข้าไปใน Table ก็ต้องใส่ให้ Vendor Table ก่อนเช่นกัน ทั้งนี้ก็เพราะ Product Table อ้างถึง V-CODE ดังนั้นถ้ายังไม่มีข้อมูลให้ V-CODE จะเกิดปัญหาเรื่อง Integrity ทันที

คำสั่งที่ใช้ในการใส่ข้อมูลเข้าไปใน Table คือ

INSERT INTO < TABLE NAME > VALUES (ATTRIBUTE1 VALUE, ATTRIBUTE2 VALUE, ... ETC)

<u>ตัวอย่าง</u>

Insert Into Vendor

VALUES (21225, 'BRYSON INC.', 'SMITH', 'WS-233-3333', 'TX', 'Y')

Insert Into Product

VALUES ("11QER/31", 'HAMMER 12 16', 20,5,800.00,25598)

เราพบว่า

- รายละเอียดของแต่ละ Row อ้างถึงที่อยู่ในวงเล็บ
- ถ้าเป็นข้อมูลแบบ Character ต้องเขียนในเครื่องหมาย " "
- ถ้าเป็นข้อมูลตัวเลขเขียนได้เลยไม่ต้องมีเครื่องหมาย ' '
- รายละเอียดของ Attribute แต่ละตัวจะคั่นด้วยเครื่องหมาย ','

Checking The Table Contents

ใช้คำสั่ง

```
SELECT *
FROM PRODUCT
```

หรือ

```
SELECT P-CODE,P-DESCRIPT,P-ONHAND,P-MIN,P-PRICE,V-CODE
FROM PRODUCT;
```

Saving The Table Contents

```
COMMIT < TABLE NAME >
```

EX

```
COMMIT PRODUCT;
```

Adding Data to The Table

ยังคงใช้คำสั่ง Insert Into... เหมือนเดิม

แล้วเมื่อเสร็จแล้ว อย่าลืมใช้คำสั่ง Commit เพื่อเก็บข้อมูลลง Disk

จากนั้นใช้คำสั่ง Select เพื่อดูรายละเอียดใน Table ว่าถูกต้องไหม

Making a Correction

ใช้คำสั่ง Update

ตัวอย่าง

```
UPDATE PRODUCT
SET P-MIN = 10, P-PRICE = 850.00
WHERE P-CODE = "11QER/31";
```

จากนั้นตรวจสอบข้อมูลว่าถูกต้องไหมด้วยคำสั่ง Select

Restore The Table Contents

ถ้าเรายังไม่ใช้คำสั่ง Commit เขียนข้อมูลลงใน Disk เราจะยังคงสามารถทำให้ Table คล้ายเป็นเหมือนเก่าได้ โดยไม่มีการเปลี่ยนแปลงใดๆ เลย นั่นคือคำสั่งเปลี่ยนแปลงรูปแบบข้อมูลไม่มีผลในการใช้งาน การ Restore ใช้คำสั่ง

```
Rollback;
```

Delete Table Rows

ใช้คำสั่ง

```
DELETE FROM PRODUCT
  WHERE P-CODE = "11QER/31";
```

หรือ

```
DELETE FROM PRODUCT
  WHERE P-MIN = 5;
```

6.3 คำถาม (Queries)

คือรูปคำสั่งที่สร้างขึ้นมาด้วย ภาษา Query เพื่อให้เกิดการนำข้อมูลออกมาจากฐานข้อมูลตามความต้องการของผู้ใช้ โดยจะใช้คำสั่ง Select เป็นหลัก โดยมีรูปแบบคือ

```
SELECT < COLUMNS >
FROM   < TABLE >
WHERE  < CONDITIONS >
```

Partial Listing of The Table

```
EX  SELECT   P-PRICE, P-DESCRIPT
    FROM     PRODUCT
    WHERE    V-CODE = 25598;
EX  SELECT   P-DESCRIPTION
    FROM     PRODUCT
    WHERE    P-PRICE <= 100.00
```

ตัวอย่าง

VENDOR (V-CODE, V-NAME, V-CONTRACT, V-PHONE, V-CITY, V-ORDER)

เรียกว่า VENDOR – Scheme (Scheme : รูปแบบที่อธิบายด้วย Text)

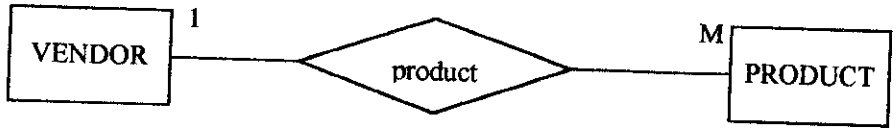
PRODUCT (P-CODE, P-DESCRIPT, P-ONHAND, P-MIN, P-PRICE, V-CODE)

เรียกว่า PRODUCT - Scheme

แต่ทั้งหมดนี้รวมเรียกว่า Schema

ทำการวิเคราะห์ ได้ว่า 1 ตัวแทนจำหน่าย สามารถจำหน่ายสินค้าได้หลายอย่าง แต่สินค้า 1 อย่าง จะมีตัวแทนได้เพียงตัวแทนเดียว

ความสัมพันธ์จะเป็นแบบ



ในตัวอย่างนี้เมื่อจะสร้าง Query จะต้องสร้าง VENDOR ก่อน เพราะมันไม่ได้อ้างอิงใคร เหมือน PRODUCT ในการใช้คำสั่ง Query นั้น ถ้าเราใช้คำสั่งว่า

```
SELECT *
FROM VENDOR;
```
} จะเป็นการบอกว่าให้เอาข้อมูลทั้งหมดใน VENDORออกมา

```
SELECT V-CODE, V-NAME
FROM   VENDOR;
```
} จะได้ข้อมูลออกมาเป็นตารางดังนี้

V-CODE	V-NAME

```
SELECT  V-CODE, V-NAME
FROM    VENDOR
WHERE   V-CODE = 22511;
```
} จะได้ข้อมูลออกมาเป็นตารางดังนี้

V-CODE	V-NAME
22511	~~~~~~

แต่ถ้าเปลี่ยนเครื่องหมายจาก = ไปเป็น <> (ไม่เท่ากับ) ผลที่ออกมาก็คือตารางที่ประกอบด้วยข้อมูลทั้งหมดทุก Row ยกเว้น Row ที่มีค่า V-CODE = 22511

```
SELECT  P-CODE, P-DESCRIPT, P-PRICE
FROM    PRODUCT
WHERE   P-PRICE > 50.00;
```

หมายเหตุ Query ที่มีการใช้เครื่องหมาย +, - , *, /, >, <, = จะเป็น Query ที่มีการนำเอา Mathematic Operator เข้ามาช่วย ส่วนใหญ่มักใช้แต่ <. , >, =

เช่น WHERE P-PRICE + 10 ; ใช้ไม่ได้ เพราะ +, -, *,/ ใช้ไม่ได้

การที่เราเรียกข้อมูลออกมาในลักษณะนี้เราเรียกว่า Data View ออกมาดู เพื่อให้เกิดการสร้างมุมมองของข้อมูลออกมาอีกชนิดตามที่ผู้ใช้ต้องการ โดย Data View ที่สร้างขึ้นมามันจะมีลักษณะเป็น Table ที่เป็น Logical Table (Table เทียม)

1. จากตัวอย่างเป็น Query ที่มีการนำ Mathematic Operator เข้ามาช่วย
2. Query ที่มีการนำเอา Logical Operator เข้ามาช่วย ได้แก่ Operator AND, OR, NOT ซึ่งมันจะให้ค่าออกมาเป็น จริง หรือ เท็จ เท่านั้น เช่น

```
SELECT  P-CODE, P-DESCRIPT, P-PRICE
FROM    PRODUCT
WHERE   P-CODE = "11QER/31"
   OR  P-CODE = "15ABC 511" ;

SELECT  P-CODE, P-DESCRIPTION
FROM    PRODUCT
WHERE   P-CODE = "11QER/31"
   AND  P-PRICE > 50.00;

SELECT  P-CODE, P-DESCRIPTION
FROM    PRODUCT
WHERE  (P-CODE = "11QER/31"  AND  P-PRICD  = 50.00)
   OR      P-CODE = "15ABC211";

SELECT  P-CODE, P-DESCRIPTION
FROM    PRODUCT
WHERE  P-CODE NOT "11QER/31"
```

ตัวดำเนินการพิเศษ (Special Operators)

Between : ใช้กำหนดขอบเขตควบคุม

Is Null : ใช้ตรวจสอบค่า Null

Like : ใช้ตรวจสอบ String คล้าย

In : ใช้ตรวจสอบค่า Attribute ว่าอยู่ในค่าของ Set ที่กำหนดให้ หรือไม่

Exists : ใช้ตรวจสอบค่า Attribute มีค่าหรือไม่

EX SELECT P-CODE, P-DESCRIPTION

FROM PRODUCT

WHERE P-PRICE BETWEEN 10.00 AND 50.00;

จะเหมือนกับ

P-PRICE >= 10.00

AND P-PRICE <= 50.00; นั่นคือระหว่าง 10.00 ถึง 50.00 ด้วย

SELECT P-CODE, P-DESCRIPTION

FROM PRODUCT

WHERE P-CODE IN ("11QER/31","15ABC211";

ซึ่งจะเหมือนกับ P-CODE = "11QER/31" OR P-CODE = "15ABD211"

SELECT P-CODE, P-DESCRIPTION

FROM PRODUCT

WHERE V-CODE IS NULL ;

ต้องการตัวที่ V-CODE เป็นค่า NULL เช่น

PRODUCT

P-CODE	P-DESCRIPTION	P-ONHAND	P-MIN	P-PRICE	V-CODE
11QER/31	~~~~~~~~~~	~~~~	~~~	~~~~	22511
15ABC211	~~~~~~~	~~~~	~~~~	~~~	

เป็น Null

```
SELECT      V-CODE, V-NAME
FROM        VENDOR
WHERE       V-CONTACT  LIKE "SMITH%"
```

% บอกว่าเมื่อมีหรือไม่มี ตัวอักษรอื่นต่อท้าย เทียบได้กับ Dummy

```
SELECT      V-CODE, V-NAME
FROM        VENDOR
WHERE       (V-CONTACT  LIKE "SMITH%")
      AND   (V-CITY  LIKE "TX");
```

```
SELECT      P-CODE, P-DESCRIPTION
FROM        PRODUCT
WHERE       P-ONHAND - P-MIN > 15 ;
```

```
SELECT      P-DESCRIPTION
FROM        PRODUCT
WHERE       P-CODE > "11QER/31" ;
```

คำสั่งจัดการข้อมูลชั้นสูง (Advanced Data Management Commands)

1. ALTER : เป็นคำสั่งที่จะนำเข้าไปแก้ไขโครงสร้างของ Table มีการใช้อยู่ 2 รูปแบบ

   ALTER MODIFY

   ALTER AND ดังนี้

```
ALTER TABLE < TABLE NAME >
MODIFY      ( < COL.NAME > < COL.CHARACTERISTICS > ) ;
หรือ        ADD ( < COL.NAME > < COL.CHARACTERISTICS > ) ;
```

EX ALTER TABLE PRODUCT

MODIFY (P-PRICE DECIMAL (9, 2)) ;

หรือ

ALTER TABLE PRODUCT

ADD (P-SALECODE CHAR (1)) ;

ส่วนการใช้ข้อมูลให้ใช้คำสั่ง UPDATE

UPDATE PRODUCT

SET P-SALECODE = '2'

WHERE P-CODE = '11QER/31' ;

UPDATE PRODUCT

SET P-SALECODE = '1.

WHERE P-CODE IN ('11QER/31', '1535-AB1', '....') ;

การ Copy ส่วนของ Table

CREATE TABLE PART (

PART-CODE VCHAR(8) NOT NULL UNIQUE,

PART-DESCRIPT VCAHR(30),

PRIMARY KEY (PART-CODE)) ;

INSERT INTO PART (PART-CODE, PART-DESCRIPT)

SELECT P-CODE, P-DESCRIPT

FROM PRODUCT ;

(WHERE .....)

ลบตาราง (Delete Table)

```
DROP TABLE <TABLE NAME>;
```

Primary and Foreign Key Designation

```
ALTER TABLE PRODUCT
ADD   PRIMARY KEY (P-CODE);

ALTER TABLE PRODUCT
ADD FOREIGN KEY (V-CODE) REFERENCES VENDOR;
```

คำถามที่ซับซ้อน (More Complex Queries)

Order ใช้สำหรับเรียงลำดับ (DESC จากมากไปหาน้อย ถ้าใช้ ORDER แล้วไม่มี DESC จะเรียงจากน้อยไปหามาก)

```
SELECT      P-CODE , DESCRIPT
FROM        PRODUCT
ORDER BY    P-CODE ;

SELECT      P-CODE , P-DESCRIPT
FROM        PRODUCT
ORDER BY    P-CODE DESC ;

SELECT      V-CODE , V-NAME , V-CONTACT
FROM        VENDOR
ORDER BY  ( V-NAME , V-CONTACT DESC ) ;
```


CT 316 (S)

```
SELECT      V-CODE , V-NAME
FROM        VENDOR
WHERE       V-NAME LIKE 'B%'
ORDER BY    V-CODE , V-NAME DESC ;
```

ความแตกต่าง (Distinct)

```
SELECT      DISTINCT V-CODE
FROM        PRODUCT ;
```

SQL Numeric Functions

COUNT : คำสั่งที่ใช้แสดงถึงจำนวนนับ

MIN : คำสั่งที่ใช้สำหรับหาค่าน้อยที่สุด

MAX : คำสั่งที่ใช้สำหรับหาค่ามากที่สุด

SUM : คำสั่งที่ใช้สำหรับหาค่ารวมทั้งหมด

AVG : คำสั่งที่ใช้สำหรับหาค่าเฉลี่ย

```
SELECT      COUNT (*)
FROM        PRODUCT
WHERE       P-ONHAND > 10 ;
```

← จะนับ Record ทั้งหมดที่มีอยู่ใน Table

```
SELECT      MIN ( P-PRICE )
FROM        PRODUCT ;
```

← จะแสดง Record ที่P-PRICE น้อยที่สุด

```
SELECT      P-CODE , P-DESCRIPT
FROM        PRODUCT
WHERE       P-PRICE = MAX ( P-PRICE ) ;
```

← จะแสดง Record ที่P-PRICE มากที่สุด

ใช้แบบนี้ดีกว่า

```
SELECT      P-CODE , P-DESCRIPT
FROM        PRODUCT
WHERE       P-PRICE = ( SELECT      MAX ( P-PRICE )
                        FROM        PRODUCT ) ;
```

```
SELECT      SUM ( P-ONHAND * P-PRICE )
FROM        PRODUCT ;
```

← จะแสดง ผลรวมของ P-PRICE ทั้งหมด

```
SELECT      AVG ( P-PRICE )
FROM        PRODUCT ;
```

หาค่าเฉลี่ย

```
SELECT      P-CODE , P-DESCRIPT
FROM        PRODUCT
WHERE       P-PRICE > ( SELECT      AVG ( P-PRICE )
                        FROM        PRODUCT )
ORDER BY    P-DESCRIPT ;
```

สร้างวิว (Create View)

```
CREATE VIEW  PRODUCT-1 AS
        SELECT      P-CODE , P-DESCRIPT , P-PRICE
        FROM        PRODUCT
        WHERE       P-PRICE >= 50.00;
        SEECT *
        FROM        PRODUCT-1 ;
```

ใช้ตารางร่วมกัน (Joining The Table)

```
SELECT      P-CODE , P-DESCRIPT , P-PRICE , V-NAME
FROM        PRODUCT , VENDOR
WHERE       V-CONTACT LIKE ( 'SMITH' )
AND         PRODUCT.V-CODE = VENDOR.V-CODE ;
```